Aalto University
School of Business

Modelling and solving logistical problems with combinatorial optimization

Case Parmatic

Bachelor's Thesis
Saska Karsi
01.06.2017
Program

Approved in the Department of Information and Service Economy xx.xx.20xx and awarded the grade

Aalto University
School of Business

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of bachelor's thesis

Author Saska Karsi

Title of thesis Modelling and solving logistical problems with combinatorial optimization: Case Parmatic

Degree Bachelor of Science

Degree programme Business Technology

Thesis advisor(s) Jyrki Wallenius, Juuso Liesiö

Year of approval 2017 **Number of pages** 22 **Language** English

Abstract

This paper deals with various logistical optimisation problems by modelling them with modified versions of, or problems related to, the Vehicle Routing Problem. The problem is modelled as an Asymmetrical Capacitated Vehicle Routing Problem with multiple vehicles, the number of vehicles being found from solving the Bin Packing Problem.

First, a linear programming formulation is constructed. Then an object-oriented programming implementation is derived from the original formulation, and implemented into a tool used to solve an empirical case from a construction company. The tool used is a Java-based application developed by the author.

Keywords Combinatorial optimization, Vehicle Routing Problem, Practical application

Table of Contents

1. Introduction

1.1 Relevance of research topic and purpose of the paper

Combinatorial optimization has been used widely to model and optimize logistical problems for a substantial amount of time (Lenstra and Rinnooy Kan, 1975). This paper discusses vehicle routing, which is of considerable importance to companies due to routing optimization's ability to mitigate transportation costs.

The purpose of this paper is to first consider the theory of combinatorial optimization and its relation to modelling and solving problems in graph theory. A solution to an empirical problem is then presented and implemented using this information.

As is often the case with modelling real-life situations as computational problems, deciding which variables can be simplified and how is extremely important.

The ultimate research question of this paper then considers finding a balance between two things; how can a vehicle routing problem be modelled accurately enough to be useful, while retaining actual solvability and computability.

The final purpose of this paper is to model the real-life situation according to the research question, and then implement the model and demonstrate functionality. Development of a possible complete real-life application is outside the scope of this paper.

A further emphasis is put on presenting the data, and the solutions, in a somewhat reader-friendly form.

2. Theoretical background

2.1. Graphs and Hamiltonian cycles in linear programming

Define $G = (V, E)$ as a graph with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and edge set $E = \{(v_i, v_j): i \neq j, v_i, v_j \in V\}$. Associate with E a cost (or distance, or time) matrix $C = (c_{ij})$, where c_{ij} is the edge weight, e.g. the cost of "travel" between v_i and v_j. When $c_{ij} = c_{ji}$ for all i, j the cost matrix, and the problem, is called *symmetric*.

If the graph is not complete, for the purposes of linear programming the value for c_{ij} in C corresponding to edge $e = (v_i, v_j)$ can be defined as a very large number. Then, if an optimal solution is found without edge e, it can be considered a Hamiltonian path. If no solution without e is found, a Hamiltonian path does not exist (Jünger et al., 1995).

A Hamiltonian cycle (Figure 1) is a cycle through a graph which visits each vertex of the graph exactly once (except the one vertex that is both the start and the end point, which is visited twice). A Hamiltonian path is a path that visits each vertex exactly once, i.e. a Hamiltonian cycle that does not connect back to the start point.

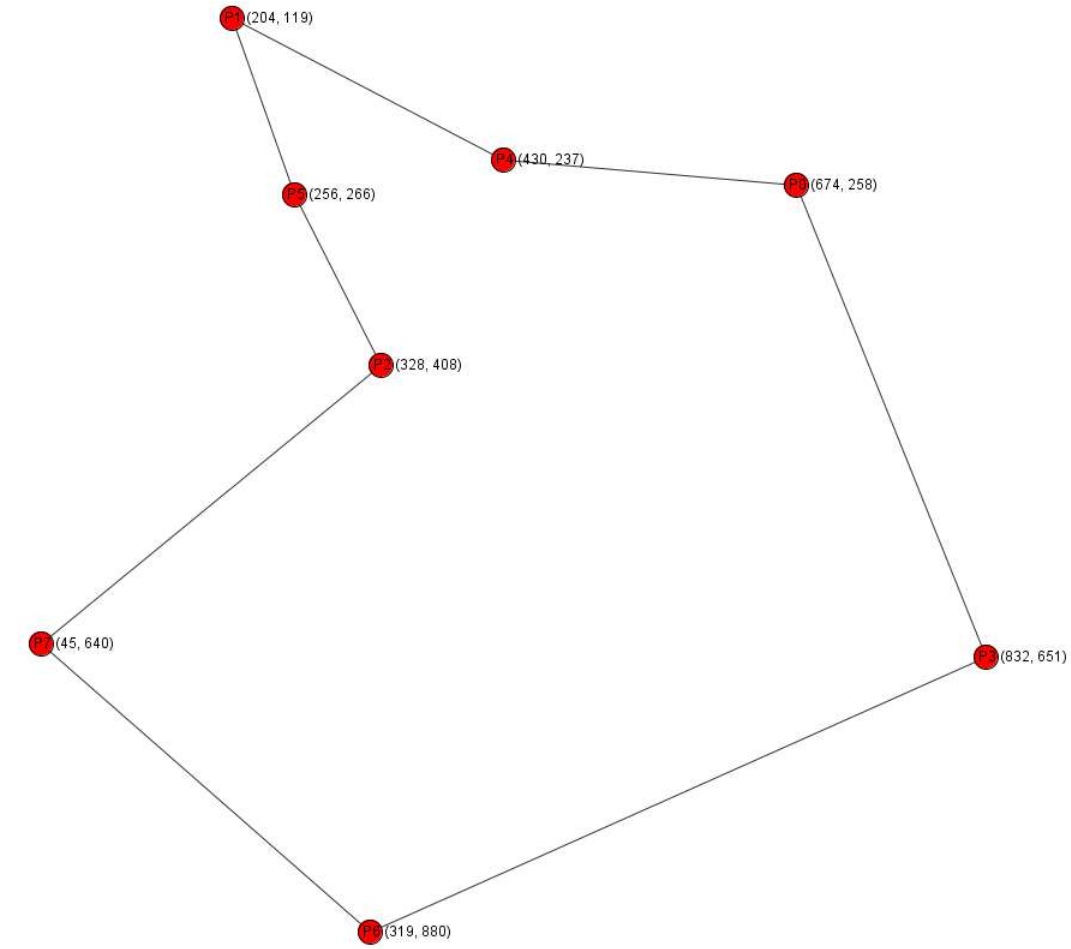

Figure 1: A Hamiltonian path (as well as the optimal solution). Vertices, or nodes, are denoted with P0...P7 with corresponding x and y coordinates to their right and the edges are drawn as lines

2.2. Graphs and the Traveling Salesman Problem

The *Traveling Salesman Problem* (henceforth TSP) is a problem asking the following question: "For a set of cities and the distances between, what's the shortest route through each city and back to the origin city, visiting each city once?" (Jünger et al., 1995).

In terms of graph theory, the problem can be formulated as finding the Hamiltonian cycle with the least possible weight from a complete weighted graph.

2.3. Other types of problems

The *Vehicle Routing Problem* (VRP) asks the same question as the TSP, but using multiple vehicles (salesmen). What is the shortest combined path for K vehicles to visit all vertices once starting from and ending to a central depot v_0? The VRP (without other constraints) is equivalent to a TSP with multiple salesmen. (Laporte, 1992).

A *Capacitated Vehicle Routing Problem* (CVRP) accounts for the limited capacity of a vehicle. Intuitively, a vehicle cannot carry an infinite amount of cargo. Thus, the demand for a given route cannot exceed the capability of the vehicle. A lower bound for the number of vehicles can be found using the *Bin Packing Problem*, which will be explained later in this section. The CRVP generalizes the Traveling Salesman Problem, which is a case of the CRVP where the capacity of the vehicle is greater than the demand of the graph and $K = 1$. (Toth and Vigo, 2002).

The *Asymmetric Capacitated Vehicle Routing Problem* is a special case of the VRP, where the cost of travel from vertex i to vertex j is not necessarily the same as the cost from j to i. In other words, the symmetry requirement $c_{ij} = c_{ji} \; \forall \; i, j$ does not hold. The problem then becomes non-Euclidian (and further non-metric), since it does not follow the rules of metric space, and the triangle equality $c_{ik} + \; c_{kj} \geq c_{ij}$ cannot be strictly held true for all i, j, k. This is the way the final empirical problem will be modelled in this paper. (Toth and Vigo, 2002).

The *Bin Packing Problem* (BPP), with a set of identical containers and a set of items, finds out what is the minimum number of containers with a constant volume that can fit the set of items, each with their own volume. The problem is computationally more complicated than it at first glance appears to be, since the number of required computational sizes increases rapidly with the size of the item set. This is also true for almost all versions of the TSP and VRP. (Fleszar and Hindi, 2002).

3. Optimizing Logistics at Parmatic

Parmatic is a small construction renovation located in Espoo. Their main logistics operator is a single driver with a van, whose responsibilities include:

- Sourcing
- Purchasing
- Delivery
- Collections
- Waste disposal
- Warehousing

The company has one main retailer for sourcing, K-Rauta Merituuli. Work sites, of course, vary with open contracts. For the purposes of the study a list from spring 2017 will be used (Table 2). The problem will, however, be formulated to account for any instance. Our problem deals mainly with the sourcing, purchasing and delivery aspects, which were seen as the most important and pressing. All other things can be outsourced or done at less busy times. The scenario is as follows: the driver is sitting in his van at the parking lot of K-Rauta Merituuli. He (for the interviewed driver was male) has orders to a set of locations. Items must be picked from the hardware store and transported to the correct construction sites. Our question is; "In which order should the driver visit the sites and resupply at the hardware store in order to minimize the amount of time, while ensuring that every construction site gets what they need, considering that the van cannot be filled over its capacity?"

To formulate the problem, some key characteristics of the empirical case must first be recognized and then modelled using objective functions and constraints. Forming computational relaxations is a secondary objective.

3.1. Formulation as a VRP

The first thing to consider is that the problem is non-Euclidean. This comes intuitively; since the cost of a trip is more than a function of the trip's length, the cost associated with edge c_{ij} in edge set E is also not the straight-line distance between v_i and v_j in vertex set V.

Furthermore, the problem is also non-metric. Because of e.g. one-way streets and intersections, it's logical to think that the cost (or time or distance) *from* and *to* a place aren't necessarily the same. Thus, the symmetry requirement $c_{ij} = c_{ji}$ of metric space is violated. By convention graph G will henceforth be defined as $G = (V, A)$ where A is a set of directed arcs instead of undirected edges.

A modified triangle equality can, however, be held true. This is to say that between two places the direct route can never be longer than an indirect route – barring unusual circumstances. The word "modified" is used to highlight that this is not a true triangle inequality, as it is *directed* and not tied to any coordinate system. Nonetheless, it's a point of interest and also comes intuitively – if one could get from place A to place B faster by visiting place C, one could merely visit place C on their way to B. This is due to the way GPS pathfinding works, as there is no real-life reason *not* to visit C. Again, this is not a true triangle inequality and cannot be used to tie the problem into metric space.

The difference in vehicular maintenance and fuel costs between routes is held negligible and the matrix (Table 1) is constructed simply using time (in minutes). These values have been gathered via Google Maps and confirmed to be accurate during an interview with a Parmatic delivery driver. The assumption will also be made that the loading or unloading time at any given construction site will stay constant regardless of the order these sites are visited in, so unloading time is excluded from the scope and is not considered in the calculations for the optimal path.

		To	To	To	To	To	To	To	To	To
		K-Rauta Merituuli	**Karamzinin koulu**	**Nöykkiölaakson koulu**	**Eestinkallion koulu**	**Mäkkylän päiväkoti**	**Toppelundin päiväkoti**	**Espoonlahden tukikohta**	**Perkkaanpuiston koulu**	**Tapiolan uimahalli**
	K-Rauta Merituuli	-	21	8	6	16	12	7	14	12
From	**Karamzinin koulu**	18	-	18	19	18	22	21	16	21
From	**Nöykkiölaakson koulu**	10	19	-	7	20	16	7	18	16
From	**Eestinkallion koulu**	6	20	7	-	18	15	10	17	15
From	**Mäkkylän päiväkoti**	15	17	18	17	-	16	17	5	11
From	**Toppelundin päiväkoti**	12	21	16	15	16	-	14	14	9
From	**Espoonlahden tukikohta**	7	20	6	8	16	12	-	14	13
From	**Perkkaanpuiston koulu**	14	15	16	15	4	15	15	-	9
From	**Tapiolan uimahalli**	13	21	16	15	11	10	15	10	-

Table 1: Asymmetric cost matrix of Parmatic supply and open construction sites. Data source: maps.google.com.

Assuming a real-world scenario such that the vehicle used has a limited capacity, it is sensible to formulate the problem as a *Capacitated Vehicle Routing Problem* or CRVP (Toth and Vigo,

2002). While the Vehicle Routing Problem is classically considered to optimize routing for a vehicle set K, K can equally be defined as a set of trips for a single vehicle. The only discernible difference is that a single vehicle cannot execute multiple trips simultaneously, which when not considering on-site waiting time does not affect the total cost of the routing. Any given trip starts at the supply point, or depot, here K-Rauta Merituuli. The depot is denoted as v_0, and without any additional constraints a single-depot Vehicle Routing Problem is equal to an m-TSP (Multiple Traveling Salesmen Problem).

A way to model the VRP as a TSP is to introduce $K - 1$ dummy clones of the depot. That is, vertices with the same coordinates and other variables as the original depot. As a result, there are K identical depots. As a curiosity, the identity of indiscernibles of metric space can be considered to be broken. Considering every visit to the depot as a separate depot is, however, mainly a computational trick and since the problem is non-metric regardless, any and all philosophical implications are outside of the scope of this paper.

It's important to make sure the arcs between the K depots cannot be traversed, and this can be done by assigning them a large positive value in the cost matrix. If W is then defined as the set of all vertices, including the dummy vertices, and continue to define V as the original set without them, the constraint can be formulated as

$$\forall\, v_i v_j, \{v_i, v_j\} \subseteq \{v_0, V \backslash W\}\colon c_{ij} = +\infty$$

which is equivalent to "if vertices i and j are both either vertex 0 or belong in the set of dummy vertices (which include v_0 as well as any vertex in W that is not in V), the corresponding edge c_{ij} in the cost matrix is positive infinity". Positive infinity can computationally be replaced with a very large number.

When interviewed, the driver said he "always tries to minimize the amount of trips to the hardware store", since "it takes a massive amount of time". This is due to the size of the store, and order-picking (although a problem solvable by this same algorithm) can, at worst, take an hour. Thus, the edge cases will not be considered even if they might exist, and K will be found by solving the *Bin Packing Problem* (BPP). This means finding out the minimum number of bins, or vans, that can supply the demand of the vertices. The problem is normally formulated to find a partition (here a K-partition of V, which means splitting V into K subgraphs) that fulfils the condition that every subset's demand is smaller than the capacity of the van, but

that is unnecessary in our case. While it would be a possible path in the CVRP, it is not necessarily optimal. One could, however, use a routing based on the partition as a lower bound to the computational solution should one wish to. It should be noted that while in a symmetric TSP or m-TSP the optimal solution will never have an edge cross another in a Hamiltonian path, in a CRVP this is possible (and when asymmetric, it is also possible within a "subtour", which will become relevant later).

3.2. Solving the Bin Packing Problem

Since it does not make sense to partially fill an order if it can be avoided (that would add a trip), the whole demand of a vertex needs to "fit" in the van.

A demand $d_i \in \mathbb{R}^+ \cup \{0\}$ is given to each vertex v_i. It's defined here as a simple non-negative real number for simplicity. Defining the demand as specific items, each with their own dimensions would provide questionable benefit (explained in the section relating to the shortcomings of the model).

A capacity $b \in \mathbb{R}^+$ is given to the van. As this is the maximum amount the van can hold at one time, the demand for a single route $d(S)$ (where $S \subseteq V$ is used to denote the demand of a subset, or a cut, of construction sites) cannot exceed b.

With these definitions (with n being the number of vertices, and noting that l is the index of the bin, or in our case van, while i is the index of a vertex in V) the BPP can be formulated as

$$\min K = \sum_{l=1}^{n} z_l$$

$$s.t.$$

$$K \geq 1 \tag{1}$$

$$\forall\, l \in \{1, \dots, n\}: z_l = \begin{cases} 1, & d_l \geq 0 \\ 0, & d_l = 0 \end{cases} \tag{2}$$

$$\forall\, l, i \in \{1, \dots, n\}: y_{il} \begin{cases} 1, & i \in l \\ 0, & i \notin l \end{cases} \tag{3}$$

$$\forall\, l \in \{1, \dots, n\}: \sum_{i=1}^{n} d_i y_{il} \leq b z_l \tag{4}$$

$$\forall\, i \in \{1, \dots, n\}: \sum_{l=1}^{n} y_{il} = 1 \tag{5}$$

The constraints work as follows: (2) specifies that z_l is one if the demand associated with van l is non-zero (if the van is required on the route). Note that K in the objective function is the sum of non-empty vans in a series of n vans (some empty, some nonempty). This follows naturally from the fact that since if a single demand is larger than the capacity, the problem becomes unsolvable. Thus, the maximum amount of vans for any solvable problem must be smaller than, or equal to, n.

Constraint (3) specifies that y_{il} is one if the demand of vertex i is associated with bin l, that is if the things that a construction site requires are packed in van l.

Constraint (4) specifies (in conjunction with the last constraint) that the sum of the demands associated with van l must be smaller than the capacity of the van. Note that if z_l is zero, nothing can be packed into the van.

Constraint (5) specifies that for all i, the sum of all y_{il} must be one. In other words an item has to be packed into exactly one van (and not zero – packing it into more than one van would require significant effort and couldn't improve the optimal solution).

3.3. Final Asymmetric Capacitated Vehicle Routing Problem Formulation

After solving for K, corresponding with the cost matrix C define $n^2 - n$ binary variables x to indicate whether an arc belongs to a route. If arc $(i,j) \in A$ is in the optimal route, x_{ij} $(i \neq j)$ takes the value of 1; otherwise, x_{ij} takes the value of zero. The values x_{ij} represent the solution graph's adjacency matrix X. The main diagonal is undefined, or defined as very large numbers.

With these characteristics in mind, the problem can be modelled as an *Asymmetric Capacitated Vehicle Routing Problem*, or ACVRP. A linear programming formulation of the problem is as follows:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{6}$$

$$s.t.$$

$$\forall\, i,j \in V: x_{ij} \in \{0,1\} \tag{7}$$

$$\forall\, j \in V \setminus \{v_0\}: \sum_{i \in V} x_{ij} = 1 \tag{8}$$

$$\forall\, i \in V \setminus \{v_o\}: \sum_{j \in V} x_{ij} = 1 \tag{9}$$

$$\forall\, S \subseteq V\ \{0, V \setminus W\}, S \neq \emptyset: \sum_{i \in S} \sum_{j \in S} x_{ij} d_j \leq b \tag{10}$$

Where the objective function, (6), is the sum of the sums of the values in the rows (or columns) of a matrix defined as the Hadamard product (Million, 2007) of matrices C and X. In other words, each value of c_{ij} is multiplied by the corresponding value x_{ij}, either one or zero, based on whether the edge appears in the graph. The sum of each value in the matrix is the total cost of the path.

Constraint (7) defines x_{ij} as either one or zero, while constraints (8) and (9) specify the degrees of the vertices in the graph: if x_{ij} can only be zero or one and the sum of each row and column in adjacency matrix X is one, it follows that each vertex has one edge entering and one edge leaving it, thus giving it a degree of 2 and applying the constraint that each vertex can only be visited once. The last constraint, (10), is the capacity constraint, specifying that any non-empty subset of the graph cannot be adjacent if its demand exceeds the capacity of the van, in other words the van cannot travel through a path if the total demand is more than it can carry.

This is a very simple formulation of the problem, with no relaxations. Implementing this as-is with or without a lower bound will take large amounts of computational time, and enforcing only these rules in an implementation would be considered a brute-force algorithm.

4. Solution approach

Solving the Bin Packing Problem is done through simple iteration. There are a number of different exact and approximate algorithms one could use that are orders of magnitude faster, but with our current problem size optimal computational efficiency is not of large concern. Solving the BPP can be conducted by defining $K = \{1, \dots, n\}$ and iterating through the ACVRP algorithm using different values of K until a feasible solution is found – thus also finding the minimum value of K.

The tool uses what is effectively a brute force algorithm – while it is technically branch-and-bound, the ordering of the subproblem queue is such that this aspect is underutilized. While ordering would be somewhat trivial to implement, the research topic of this paper does not

concern computational efficiency, and the problems this algorithm is used with in this scope are not large enough to present a problem.

The tool uses a simple subproblem queue, from which it "pops" the most recently added subproblem (here a tour). When a tour is retrieved, it checks whether that tour is complete. If the tour is not complete, the code iteratively adds to the queue every feasible continuation of the tour with a new vertex. If the tour is complete, it checks the tour against the shortest one found.

For the purposes of object-oriented programming, the capacity constraints can be equally formulated as

$$b_i = \begin{cases} b_{i-1} - d_i, v_i \in V \backslash \{v_0\} \\ 1, v_i \in W \backslash V \cup \{v_o\} \end{cases}$$

$$s.t.$$

$$b_i \geq 0$$

where b_i is the capacity of then van at vertex v_i. This models a situation where if the visited vertex is a depot (either v_0 or one of the dummy vertices) the capacity of the car will be reset back to 1, while if the vertex visited is one with a non-negative demand, the capacity will be the previous capacity minus the demand. The capacity cannot drop below zero.

One of the largest benefits of this type of object-oriented programming (where the van "traverses" the vertices instead of maximising a simple objective function, thus recognising the degree of a vertex as two when only one edge is traversed both ways) is the ability to do single-vertex routes. This is important in modelling a real-world scenario where a driver can leave a depot, make a single delivery, and go straight back to the depot. This is necessary e.g. when the demand of a single construction site fills the whole capacity of the van.

5. Results

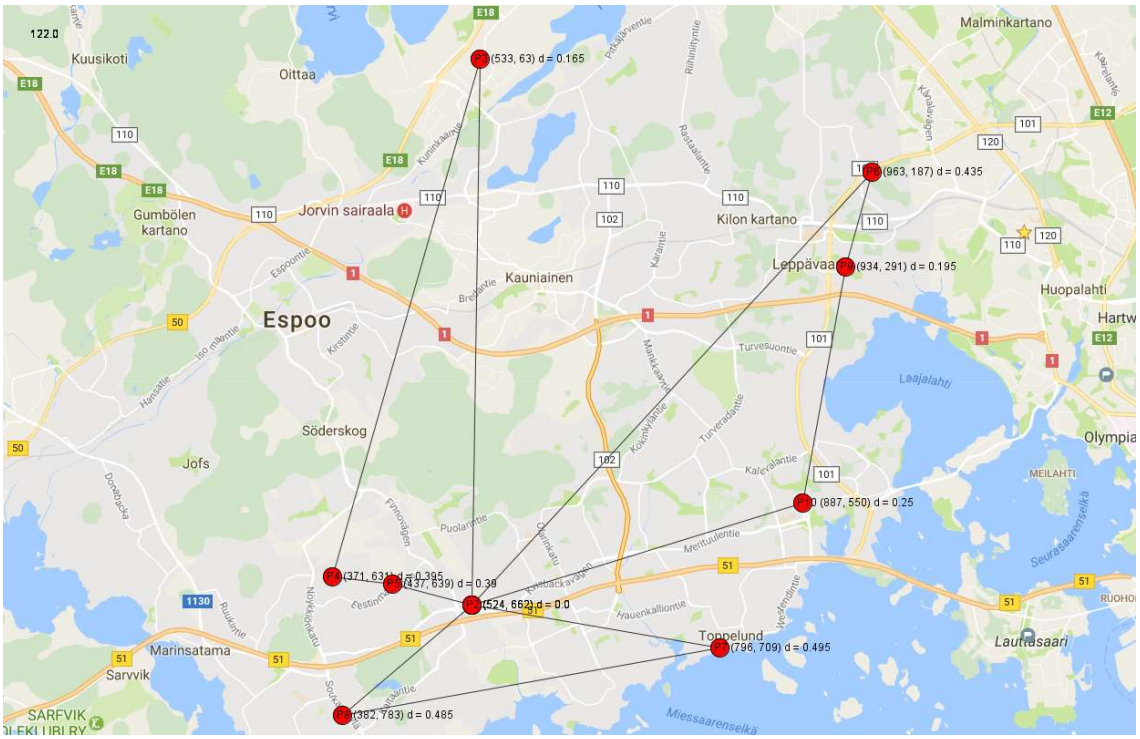

Figure 2: Program output visualisation. Background image: maps.google.com

This is the output visualisation for a $K = 3$ problem. It should be noted that while the arcs drawn are purely Euclidian straight line distances between the vertices, they are purely for demonstrating the path. The cost matrix used for the calculational edges can be found in *Figure 2*. The visualisation lacks directional edges, but the directional path can be read from the text output. The program prints the vertices in order of traversal and the values of P_i in the program output correspond with the values in the vertices of the visualization and the vertex set V.

Program output:

```
P0 (524, 662) d = 0.0
P5 (437, 639) d = 0.39
P4 (371, 631) d = 0.395
P3 (533, 63) d = 0.165
P1 (524, 662) d = 0.0
P8 (382, 783) d = 0.485
P7 (796, 709) d = 0.495
P2 (524, 662) d = 0.0
P10 (887, 550) d = 0.25
P9 (934, 291) d = 0.195
P6 (963, 187) d = 0.435
P0 (524, 662) d = 0.0
122.0
```

This is the directed path. It contains a vertex index in the form P_i, the coordinates (in relation to the program drawing board – not a real-life coordinate system) and a value for the demand of the vertex, d. The last number is the total cost of the route. The sum of all demands in this example is 2.8 and the program found a solution where $K = K_{min} = 3$. Three depot vertices are then defined first, here P_0, P_1 and P_2 (bolded in the text output and drawn in the same coordinates on the graph in order – thus in they are all at the circle marked P_2 in the visualization). None of the subtours between depots are larger than $b = 1$.

The route can either be read from the visualization or the list of construction sites (either a row or column in Table 1, noting that the depot, K-Rauta Merituuli, has a final index of P_2. In this particular instance, the driver should travel from the depot, drive the route Eestinkallionkoulu-Nöykkiölaakson koulu-Karamzininkoulu, resupply at the depot, drive the route Espoonlahden tukikohta-Toppelundin Päiväkoti, resupply once more, drive the route Tapiolan uimahalli-Perkkaanpuiston Koulu-Mäkkylän päiväkoti and finally return to the depot. Note that the order the *routes* are taken in is irrelevant, i.e. as long as the sites in a route are taken in the correct order, the routes themselves are interchangeable. The total length of all three routes, the value of the objective function, is 122.0 minutes.

6. Discussion

6.1. Simplifications of the model

The model is, like all models, a simplification of the real-world situation. Some of these simplifications are considered here.

Firstly, the model does not consider waiting time. The cost of late delivery can be immense, especially in this type of construction site setting where a crew of five to ten people can be waiting for their supplies with no work capability whatsoever. The cost of lost man hours can be substantial. This could be solved with simple prioritisation, or by including it in the actual objective function. With proper planning, however, these situations are rare, and were thus excluded to make constructing an exact (rather than a heuristic) algorithm simpler.

Second, the capacity and demand are given as simple real numbers between zero and one. This is a necessary simplification with little drawbacks – constructing an exact algorithm for

what can and cannot fit within a van from a hardware store catalogue of tens of thousands of items is not only near impossible, but also a futile effort. The same exact algorithm would also need to be followed by whoever was loading the van, and that does not reflect a real-life scenario. In the real world construction materials can also vary from perfect cubes to sheets of $3200mm \times 1200mm \times 6mm$ to massively long timber. Multiple variables, like fragility and transit orientation, would also need to be considered. The core problem, then, is whether two distinct sets of items can feasibly be hauled together by the same vehicle.

6.2. Practical application

The driver, when presented the tool and interviewed a second time, said that it has potential to considerably benefit the operation and flow of Parmatic's limited logistical capability, particularly material supply. Particular interest was shown in the capability of the software to split the construction sites to multiple routes, since it is possible Parmatic will be hiring one or more drivers in addition to the current one. As shown previously, the algorithm will work identically whether considering one van and multiple trips, or multiple vans each doing a single trip.

Before a real-life application can be implemented, however, a few shortcomings need to be addressed.

Firstly, data input is currently painfully slow and it is hard to gain much from the model and its implementation in its current state. The scope of this paper is, however, a single instance. Automatically querying an outside map API (application programming interface) for travel durations in real-time would be trivial. The list of vertices could then equally trivially be transmitted to the same API to get pre-routed site-by-site GPS navigation.

The second problem is less fatal and was suggested by the driver. Currently, the situation is modelled in a way that requires the route to start at the depot. This is not always the case, and it would be helpful to be able to start routing from anywhere. A possible method for creating a routing based on the current position of the vehicle is to read current GPS coordinates through an API and create the first depot in the algorithm using those coordinates in creating the cost matrix.

Any further required depots would then be given the coordinates of K-Rauta Merituuli, as before.

Both of these shortcomings are possible venues for further research, the second being more relevant to the same principles already applied in this paper. The first one is largely a question of interfacing.

7. Conclusions

This has been a study into how combinatorial optimization can be used to model and solve logistical problems. Doing this is often deceptively complex. A relatively accurate model of the core problem was formulated with an acceptably low number of variables. This model was then implemented and used to solve a problem instance to demonstrate functionality. The algorithm worked as expected, producing the optimal result with the given graph, cost matrix and set of demands.

Computationally, with modern computers, solving instances of this size is practical even using brute force with no significant relaxations (even though all optimization problems considered in the paper are computationally extremely difficult and resource-consuming – the technical term is NP-Hard). Larger problems, or similar problems in larger volumes, could require some optimization of the algorithm. For practical applications the algorithm would also most likely run on a mobile platform or on a server (few drivers carry a computer around).

Real life applications would also require connectivity with a map API to be particularly useful. Implementing this connectivity would, however, be relatively trivial. A way to add to the functionality of the application would be the ability to start vehicle routing from any given point on a map instead of solely the depot.

Appendix 1: Java Code

The appendix contains only the core methods for minimizing the objective function.

```
public void optimize() {
		minTour = new Tour();
		double minTourLength = Double.MAX_VALUE;

		tours = new LinkedList<Tour>();

		Tour t = new Tour();
		t.addVertex(vertices.get(0));
		tours.add(t);

		while (!tours.isEmpty()) {
			t = tours.pop();

			if (getTourBound(t) >= minTourLength) {
				continue;
			}

			// if tour is hamiltonian path, add vertex 0
			if (t.getVertexAmount() == n) {
				t.addVertex(vertices.get(0));
			}

			if (t.isComplete() && t.getLength() < minTourLength) {
				minTour = t;
				minTourLength = t.getLength();
				continue;
			}

			double capacity = t.getCap();
			for (int i = 1; i < n; i++) {
				Vertex v = vertices.get(i);
				if (v.getD() > capacity) {
					continue;
				}

				// if tour doesn't have vertex v, add v
				if (!t.containsVertex(v)) {
					LinkedList<Vertex> l = new LinkedList<Vertex>();
					l.addAll(t.getVertexList());
					l.add(v);

					tours.add(new Tour(l));
				}
			}
		}

		System.out.println(minTour.toString());
		runCount++;
		avg += (minTour.getLength() - avg) / runCount;

		System.out.println("Average: " + avg);
	}
```

```
private double getTourBound(Tour t) {
	// Get current tour length
	double bound = t.getLength();
	double min = Double.MAX_VALUE;

	// distanceMatrix size is always n*n so laziness in the iterator.
	// t.getVertexAmount() returns
	// the amount of vertices in the tour so the iterator adds the
	// minimum amount to t.getLength(), equaling the bound.
	for (int i = t.getVertexAmount(); i < n; i++) {
		for (int j = 0; j < n; j++) {
			double d = distanceMatrix[i][j];
			if (d < min) {
				min = d;
			}
		}
		bound += min;
	}

	return bound;
}
```

Appendix 2: Notation used in this paper

G	Graph with vertex set and either an edge set or an arc set
V	Vertex set
v_i	Vertex i with v_0 being the depot
E	Edge set
e	Edge
C	Cost matrix
c_{ij}	Cost between vertices i and j in C
K	Number of vehicles or trips
A	Arc set
W	Vertex set with dummy vertices
S	Subset of V
d_i	demand at vertex i
b	capacity of van
X	Adjacency matrix of the solution graph

References

Fleszar, K., Hindi, S.: New heuristics for one-dimensional bin-packing. Computers & Operations Research 29 (2002) 821-839

Jünger M., Reinelt G., Rinaldi G.: The Traveling Salesman Problem. Handbooks in OR & MS, VoL 7, 1995.

Laporte, G.: The Vehicle Routing Problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59 (1992) 345-358

Lenstra J.K., Rinnooy Kan A. H. G.: Some Simple Applications of the Travelling Salesman Problem. Opl. Res. Q. Vol. 26, 1975. 717-733

Million, Elizabeth. The Hadamard Product. April 12, 2007

Toth P., Vigo D.: Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics 123, 2002. 487-512

Data sources

maps.google.com